Binary Vision Algorithms in Java™

R. Mukundan

Faculty of Information Technology,
Multimedia University, 75450 Melaka, Malaysia.
mukund@unitele.edu.my

Abstract

The capabilities of Java as a highly portable, multi-threaded general purpose programming language can be effectively utilized to provide an efficient framework for learning, coding, visualizing and demonstrating the fundamental concepts behind binary vision algorithms. Methods whose implementations are described in this paper include connected component labeling, boundary following algorithm, image feature extraction, and image thinning algorithm. The concept of a pseudo-screen in introduced to generate the display of a scaled pixel grid for a highly magnified view of the pixel level operations and the intermediate stages in recursive computing. The graphics and the user interface classes of Java together with the thread class are used to create methods necessary for interactive input and update of images and also for rendering the output on any Java enabled browser.

Keywords: *Binary vision, Java programming, web based courseware, image processing.*

1 Introduction

Binary vision algorithms form one of the core classes containing important procedures in computer vision. This paper discusses the implementation of these algorithms in Java programming language. The motivation for this work is the increasing use of the Internet in the learning environment as well as the capabilities of Java as a highly portable general purpose programming language with built-in classes for graphics and user interface functions [1]. The internal working of binary vision algorithms such as the recursive computation of pixel values can be clearly visualized on a graphics display running on a Java thread [2]. The user interface classes can additionally provide the mechanism for interactive menu selection, data input, image input and image update, on a separate thread.

Since Java applets run inside a web browser, it is the most suited language in a distance learning environment where students access notes and embedded applications through the Internet. The similarity of the lexical structure of Java to C, and the ease with which the Java development environment can be created, are factors that are helpful in teaching courses and developing successful projects involving graphics and image processing algorithms.

The vision algorithms discussed in this paper are recursive connected component labeling, boundary following, and image thinning. The concept of a pseudo screen is introduced below, which is designed to cater to most of the common pixel level operations, and to provide a generalized framework where the process details can be very conveniently visualized. A schematic diagram showing the major components and the flow of control and data between the components in the above framework consisting of the vision algorithm, the pseudo screen, user interfaces, graphics functions and threads is given in Fig. 1. This paper also presents the program code in Java, for the pseudo-screen class (which contains the primary display related functions) and the vision algorithms mentioned above.

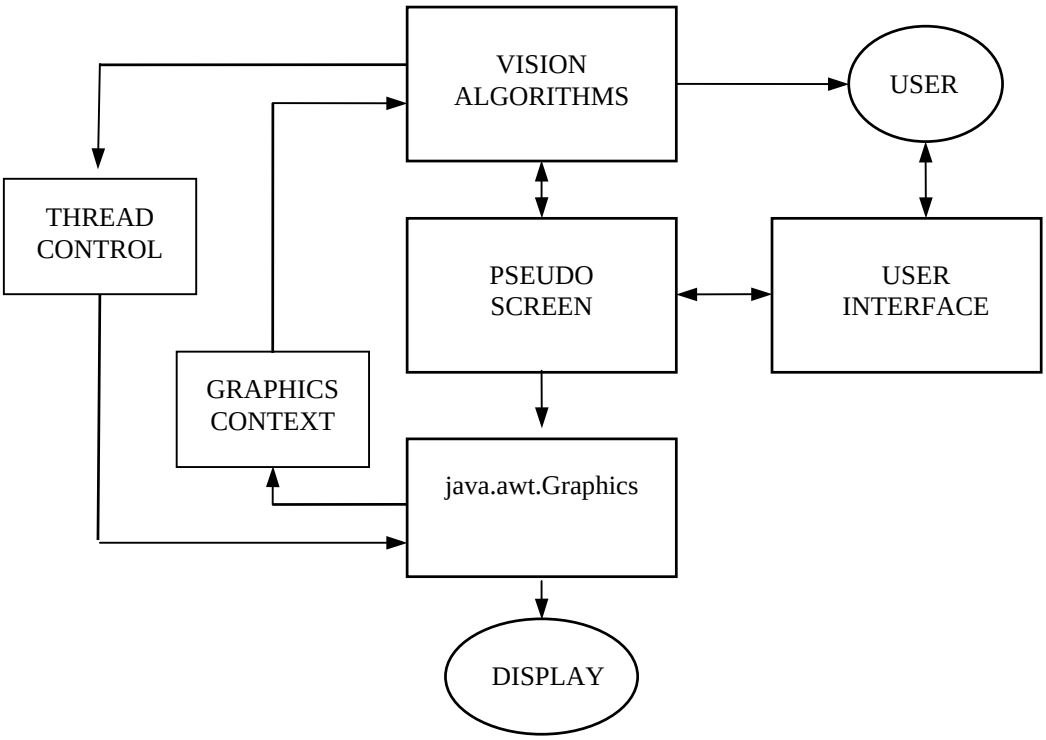

Fig. 1. Block schematic of the Java based framework for implementing vision algorithms.

2 Pseudo screen

The pseudo screen is nothing but a graphics display of the pixel array, where every pixel in the image has a much larger representation in terms of a square or a circular region. Viewed as a data structure, the pseudo screen has a frame buffer and a set of associated operations to be carried out at the pixel level such as setting a pixel, retrieving a pixel value, and clearing a pixel. A schematic of the functions of the pseudo screen class is given in Fig. 2.

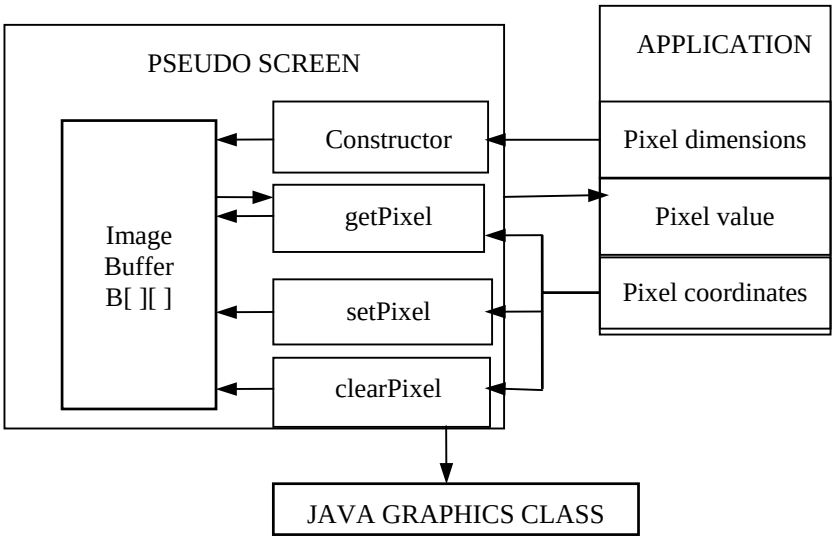

Fig. 2. Block schematic of the pseudo screen class and its components.

These basic functions are important in any vision algorithm for the manipulation of images, and particularly useful when similar functions do not exist in the Java graphics class. The Java code for the class is in Fig. 3.

```
import java.awt.*;
public class PseudoScreen {
    private Graphics g;
    private int[][] B;
    int wid,hgt;

    PseudoScreen(int w, int h){       //class constructor
      B=new int[w][h];                //set up frame buffer
      wid=w; hgt=h;                   //pixel dimensions
    }
    public void Screen(Graphics g, boolean first){
      this.g=g;                       //get graphics context
      if(first) grid();               //draw pixel grid
    }
    private void grid(){              //draw pixel array
      g.setColor(Color.white);
      g.fillRect(0,0,wid*10,hgt*10);
      g.setColor(Color.red);
      for(int x=0; x<wid*10; x+=10)
        for(int y=0; y<hgt*10; y+=10)
          g.drawRect(x,y,10,10);
    }
    public void setPixel(int i, int j, int k){
      Color[] cls={Color.white, Color.blue, Color.green,
                   Color.cyan, Color.red, Color.magenta,
                   Color.yellow, Color.orange, Color.pink,
                   Color.lightGray};
      if((i<0)||(i>wid-1)||(j<0)||(j>hgt-1)) return;
      if((k<0)||(k>9)) return;
      B[i][j]=k;
      g.setColor(cls[k]);
      g.fillRect(10*i,10*j,10,10);    //set a pixel
    }
    public int getPixel(int i, int j){
      if((i<0)||(i>wid-1)||(j<0)||(j>hgt-1)) return -1;
      return B[i][j];                 //return a pixel value
    }
    public void clearPixel(int i, int j){
      if((i<0)||(i>wid-1)||(j<0)||(j>hgt-1)) return;
      B[i][j]=0;                      //clear a pixel
      g.setColor(Color.white); g.fillRect(10*i,10*j,10,10);
      g.setColor(Color.red);   g.drawRect(10*i,10*j,10,10);
    }
}
```

Fig. 3. Java code for the pseudo screen class.

3 Vision algorithms

In this section, we discuss four important binary vision algorithms viz., the connected component labeling algorithm, the image feature extraction algorithm, the boundary following algorithm and the thinning algorithm. In all these algorithms we use 8-connectedness for foreground pixels and 4-connectedness for the background pixels. The connected component labeling algorithm uses the well known recursive procedure to assign a label (or a color value) to all 8-connected foreground pixels of a component [3]. The Java code for the procedure is given in Fig. 4.

```
void setLabels(){
   int m=2;                  //scr is a pseudo screen object
    for(int y=0; y<scr.hgt; y++)
      for(int x=0; x<scr.wid; x++)
        if(scr.getPixel(x,y)==1) compLabel(x,y,m++);
}
void compLabel(int i, int j,int m){
  if(scr.getPixel(i,j)==1){
      scr.setPixel(i,j,m);              //assign label
      slow();                           //thread delay
      compLabel(i-1,j-1,m); compLabel(i-1,j,m);
      compLabel(i-1,j+1,m); compLabel(i,j-1,m);
      compLabel(i,j+1,m);   compLabel(i+1,j-1,m);
      compLabel(i+1,j,m);   compLabel(i+1,j+1,m);
  }
}
```

Fig. 4. Java method for connected component labeling.

The component labeling algorithm given above can be further modified to output additional information such as the total number of components in a binary image and the number of pixels in each component (component area) . The image feature extraction algorithm can then determine the shape features of each component using geometric moments [3]. A block diagram showing the implementation aspects of this procedure is in Fig. 5.

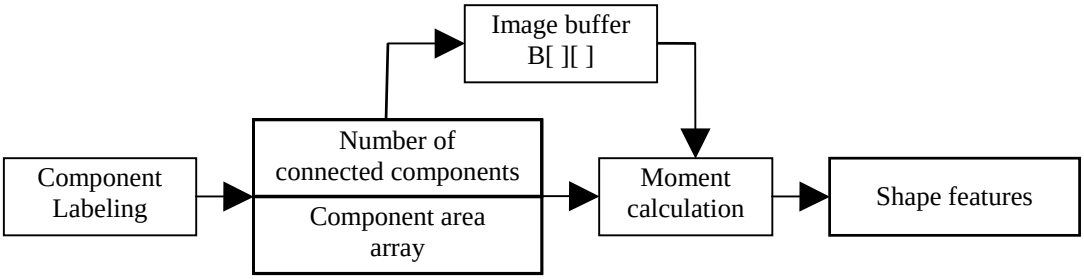

Fig. 5. Procedure for image feature extraction.

```
void getBound(int i, int j){       //initial boundary pixel
  int p, inew, jnew, k=0, i0=i, j0=j;
  int[] ioff={-1,-1,0,1,1,1,0,-1};     //nbd offsets
  int[] joff={0,-1,-1,-1,0,1,1,1};
  int[] nbd={6,0,0,2,2,4,4,6};       //nbd index mapping
  do{                                //repeat along boundary
    for(int n=0; n<8; n++){          //search in nbd
      p=k+n;
      if(p>7) p-=8;
      inew=i+ioff[p];  jnew=j+joff[p];
      if(scr.getPixel(inew,jnew)!=0){
        k=p-1;
        if(k<0) k=7;
        k=nbd[k];
        i=inew; j=jnew;
        scr.setPixel(i,j,2); slow(); break;
      }
    }
  }while((i!=i0)||(j!=j0));
}
```

Fig. 6. Java code for boundary following algorithm.

The boundary following algorithm given by the Java code in Fig. 6, tracks the boundary starting from an edge pixel of a connected component, following it in the clockwise direction until the boundary closure condition is satisfied by arriving back at the starting point. A sample input image and the corresponding output of the boundary following algorithm are shown in Fig. 7.

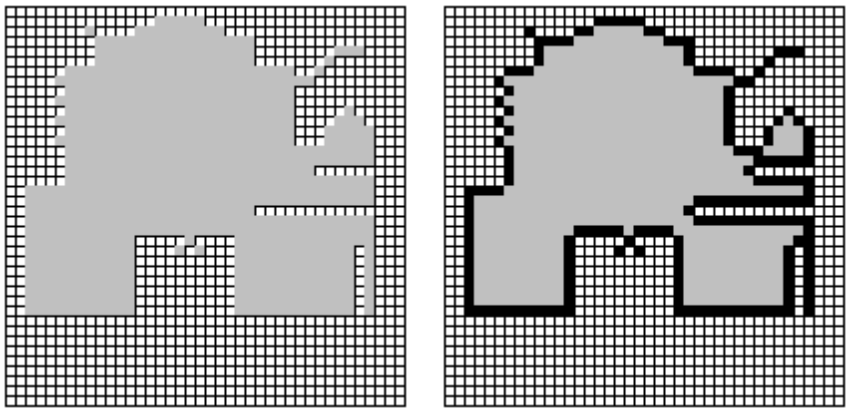

Fig. 7. An input image and the corresponding output of the boundary following algorithm.

The thinning algorithm is used for skeletonization of binary images with applications in the areas of shape representation and matching. The two pass thinning process uses successive deletion of boundary pixels until a single 8-connected one pixel wide component is obtained, which simultaneously maintains endline locations and approximates the medial lines of the original image. To save space, only the primary method containing the crux of the algorithm is given in Fig. 8. The method uses two additional functions "nbors" which returns the total number of 1-pixels in an 8-connected neighborhood of the current pixel, and "cindex" which returns the crossing index of the current pixel [4].

```
void makeThin(){
  int nb;
  boolean flag;
  do{
     flag=false;
     for(int y=0; y<scr.hgt; y++)    //Pass-1
       for(int x=0; x<scr.wid; x++)
         if(scr.getPixel(x,y)==1){
           nb=nbors(x,y);            //neighboring pixels
           if((nb>2)&&(nb<7))
             if(cindex(x,y)==1)      //crossing index
                   scr.setPixel(x,y,2);  //Mark boundary
         }

     for(int y=0; y<scr.hgt; y++)    //Pass-2
       for(int x=0; x<scr.wid; x++)
         if(scr.getPixel(x,y)==2){
            scr.clearPixel(x,y); flag=true; slow(); //Delete
         }

   }while(flag);
}
```

Fig. 8. Java code for thinning algorithm.

A sample input image and the corresponding output of the thinning algorithm are given in Fig. 9.

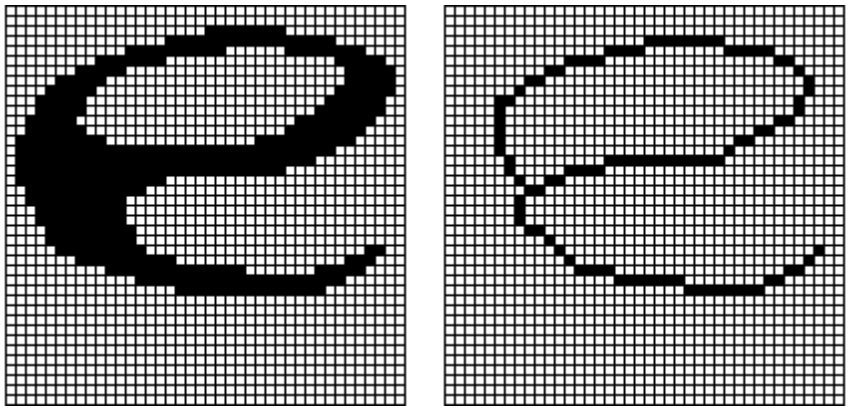

Fig. 9. An input image and the corresponding output of the thinnning algorithm.

4 Conclusions

Java provides a good medium for the implementation of vision algorithms, with its supporting classes for graphics, user interfaces, and threads. Being the language for the Internet, it is even more suitable for developing web based courseware. The work presented in this paper details some of the important binary vision algorithms in the framework of Java classes, and is intended to stimulate further research towards the development of a comprehensive set of class libraries for more advanced algorithms in image processing and computer vision.

5 Acknowledgments

The author is grateful to Prof. Lee Poh Aun (Dean of the Faculty of Information Technology, Multimedia University) for his encouragement and suggestions.

References

[1] P. Chan and R. Lee: *The Java Class Libraries*. Addison-Wesley, Massachusetts (1998).

[2] S. Oaks and H. Wong: *Java Threads*. O'Reilly (1997).

[3] R. Jain, R. Kasturi and B. Schunck: *Machine Vision*. McGraw-Hill (1995).

[4] J.R. Parker: *Practical Computer Vision Using C*. John Wiley & Sons (1994).